УДК 004.056
DOI: 10.32626/2308-5916.2024-25.22-36

М. Ю. Мітіков, аспірант,
Н. А. Гук, д-р техн. наук, професор

Дніпровський національний університет
імені Олеся Гончара, м. Дніпро

ДОСЛІДЖЕННЯ ПРОБЛЕМ ШВИДКОДІЇ ПРОГРАМНИХ ДОДАТКІВ

У статті досліджуються методи оптимізації швидкодії програмних додатків з метою визначити найбільш ефективні комбінації внутрішніх та зовнішніх чинників, що дозволяють максимізувати цільову функцію. Описано узагальнену математичну модель, яка складається з основних чинників, від яких залежить швидкодія програмного забезпечення, зокрема час виконання обчислень, кількість операцій введення-виведення, кількість обчислювальних операцій, складність алгоритму, обсяг оброблюваних даних, застосування принципів паралелізму, архітектура апаратної та програмної платформи, а також ефективність програмного коду. Зазначено важливість застосування спеціалізованих бібліотек та інструментів для прискорення обчислювальних процесів, що є критично важливим для досягнення високої продуктивності сучасних програмних систем.

Здійснено програмну реалізацію розроблених підходів, що дозволяє оцінити ефективність запропонованих методів на практиці. Розроблено відповідне програмне забезпечення, за допомогою якого здійснено аналіз впливу різних факторів на швидкодію з урахуванням специфіки конкретних завдань та середовища виконання. Результати тестування продемонстрували значний потенціал для покращення продуктивності за рахунок оптимізації як на рівні програмного коду, так і на рівні апаратної архітектури.

Особливу увагу приділено дослідженню роботи з пам'яттю, розглянуто потенційні виклики, які негативно впливають на швидкодію. Наведено необхідність використання системи кешування, та уникнення дублювання незмінної інформації. Знайдені сценарії не залежать від конкретної реалізації, і тому можуть бути додані в розроблювану рекомендаційну систему.

Дослідження має практичне значення, оскільки пропонує комплексні рішення для оптимізації швидкодії програмних систем, які можуть бути використані в промислових високонавантажених середовищах. Подальші дослідження будуть спрямовані на розширення функціональних можливостей рекомендаційної системи, інтеграцію більш складних моделей оптимізації, а також на проведення широкомасштабних обчислювальних експериментів для підтвердження отриманих результатів у реальних умовах.

Ключові слова: *швидкодія, програмні додатки, математичні моделі, методи оптимізації, програмна реалізація, рекомендаційна система, дерево рішень.*

Вступ. Швидкодія програмних додатків є важливим аспектом сучасного програмування [1], оскільки вона безпосередньо впливає на час отримання результатів розрахунків, що є важливим для користувачів та бізнесу. Із зростанням об'єму даних, складності алгоритмів та вимог до продуктивності, розробники стикаються з проблемами, пов'язаними з ефективністю роботи програмного забезпечення. Вирішення цих проблем є важливим завданням, що забезпечує задоволення користувачів, збільшує конкурентоспроможність та знижує витрати [2].

Одним із ключових аспектів в дослідженні проблем швидкодії програмних додатків є застосування математичних методів і підходів. Прикладна математика виступає як потужний інструмент для аналізу, моделювання та розв’язання задач, пов'язаних з продуктивністю програмного забезпечення.

Метою цієї статті є дослідження проблем швидкодії програмних додатків з використанням математичних методів та аналіз впливу чинників різного походження на ефективність роботи програмного забезпечення.

В дослідженні [3] розглянуто проблему оптимізації швидкодії та відмовостійкості програмних додатків. Наведено різні архітектурні рішення, що сприяють оптимізації програмних додатків з метою поліпшення їх швидкодії та відмовостійкості. Дослідниками проаналізовано вплив архітектури програмних додатків, що визначається на етапі їх проектування, на якість програмного забезпечення, розроблено метод визначення пріоритетів оптимізації програмних проектів за допомогою побудови діаграм. Встановлено, що найбільш поширеними та прийнятними для покращення швидкодії є розбиття великих команд на менші та проведення рефакторингу з метою видалення надлишкових команд, а також специфічні методи, пов’язані зі вставкою програмного коду на мові низькорівневого програмування.

В роботі [4] досліджуються різні підходи до проектування вебдодатків з метою оптимізації їх швидкодії. У своїй роботі автори наводять аналіз предметної області, проводять порівняльний аналіз підходів до проектування веборієнтованих додатків та формують критерії їх ефективності. За результатами дослідження визначено підходи до розробки вебдодатків, обрано інструменти для розробки тестового вебдодатку, а також висвітлено процес його проєктування на основі попередньо визначених підходів та тестування його ефективності.

Автор роботи [5] аналізує питання підвищення продуктивності вебдодатків та акцентує увагу на актуальності теми, оскільки вебдодатки є

основним інструментом для реалізації інформаційних послуг та електронного бізнесу. В роботі здійснюється порівняння можливостей вебдодатків та десктопних додатків, а також наводяться переваги застосування вебтехнологій. Описуються основні фактори, що обмежують продуктивність: технічні характеристики середовища, кількість користувачів та швидкість обслуговування запитів. Розглядаються вплив обсягу оперативної пам'яті, параметрів каналу зв'язку та характеристик жорстких дисків на швидкодію. Наводяться основні методи підвищення продуктивності з боку серверу, зокрема стиснення даних, різні види кешування, використання FrontEnd-BackEnd архітектури, балансування навантаження. Підкреслюється важливість вибору вебсервера та технологій розробки, що сприяє підвищенню продуктивності.

Швидкодія є однією з визначальних вимог до будь-якого програмного додатку, оскільки користувачі сподіваються отримувати негайну реакцію на свої дії. Зі збільшенням функціональності та обсягів оброблюваних даних досягнення необхідного рівня швидкості роботи програмного забезпечення стає все більш складною задачею.

Недостатньо жорсткі вимоги до швидкодії [6] можуть призвести до комерційних збитків та до втрати користувачів. Тому проблема покращення швидкодії програмних продуктів є актуальною та потребує досліджень.

З огляду літератури можна бачити, що наявні дослідження враховують лише окремі фактори, які впливають на швидкодію програмних додатків, та не розглядають взаємозв'язок внутрішніх та зовнішніх факторів.

Метою цього дослідження є розробка науково обґрунтованих рекомендацій щодо архітектурних, алгоритмічних та апаратних чинників, які сприятимуть підвищенню швидкодії програмних додатків.

Основний науковий внесок цієї роботи полягає в тому, що вона поєднує теоретичний підхід, моделі та методи прикладної математики з практичними вимірами швидкодії програмного забезпечення. Результати досліджень, представлені в цій статті закладають основу подальших досліджень.

Метою дослідження є аналіз чинників, що впливають на швидкодію програмних додатків, розроблення математичної моделі досліджуваного процесу, оцінка існуючих методів та підходів до оптимізації, програмна реалізація запропонованих підходів для перевірки ефективності розроблених рішень та узагальнення отриманих даних для формування рекомендацій.

Результати дослідження дозволять підвищити якість та продуктивність програмних додатків, що в свою чергу сприятиме задоволенню потреб користувачів, зменшенню кількості розрахункових потужностей і використаної енергії.

Постановка задачі. Нехай *S** – цільова (оптимальна) швидкість (швидкодія) програмного додатку, яку прагнемо досягти.

У загальному вигляді швидкодія програмного додатку може бути зображена у вигляді цільової функції виду:

$$S = f\left(C_1, C_2, C_3, ..., C_n\right), \quad (1)$$

де S – швидкодія програмного додатку, C_1, C_2, C_3,..., C_n – основні чинники, що впливають на швидкодію. Функція f визначає природу залежності між швидкодією та цими чинниками. Необхідно на множині допустимих розв'язків **Ω** знайти такий вектор $C^* = \left(C_1^*, C_2^*, C_3^*, ..., C_n^*\right)$, – точку в n-мірному евклідовому просторі R^n за наявності обмежень $g(C_1, ..., C_n) = b$ на множину допустимих розв'язків **Ω**, яка надає екстремуму цільовій функції, тобто:

$$S^* = f\left(C^*\right) = \underset{c \in \Omega}{\text{extr}}\, f\left(C\right).$$

Аналіз основних чинників, що впливають на швидкодію. До основних чинників, які впливають на швидкодію, належать обсяг інформації з якою працює програмний додаток, обсяг обчислювальної роботи, та характеристики апаратної частини, що виконує обчислювальну роботу. Чинники між собою взаємопов'язані, при збільшені кількості оброблювальних даних, зростає необхідність у збільшені оперативної пам'яті для зображення даних у пам'яті для подальшої обробки. R – кількість операцій, пов'язаних з введенням і виведенням даних, які виконуються під час роботи програмного додатку. Прикладом операції введення/виведення може бути запит до мережевого ресурсу, бази даних, або доступ до файлової системи (локальної чи розподіленої). P – кількість обчислювальних операцій, які виконуються в програмному додатку, таких як арифметичні та логічні операції. N – обсяг даних, з якими працює програмний додаток. A – характеристика апаратної та програмної архітектури, на якій працює додаток, включаючи тип процесора, пам'ять, операційну систему та інші компоненти.

CPU — зважена оцінка якості процесора, що враховує його ключові характеристики:

- **Кількість ядер** визначає кількість паралельних/незалежних обчислень які можуть одночасно виконуватись.
- **Тактова частота** визначає кількість операцій, які процесор може виконати за одиницю часу. Деякі моделі енерго-ефективних процесорів мають декілька ядер з різною тактовою частотою і різним рівнем споживання енергії. При малих навантаженнях на систему, активним є енергоефективне ядро.
- **Розмір кеш-пам'яті** визначає об'єм даних до яких є швидкий доступ. Доступ до даних в оперативній пам'яті займає у 10 разів більше часу ніж доступ до даних в процесорному кеші [2].

- **Наявність новітніх команд процесора** - підтримка таких технологій, як SIMD (Single Instruction, Multiple Data), дозволяє процесору виконувати одну інструкцію над набором даних одночасно. Наприклад, в процесорах без підтримки SIMD, операція сумування 128 чисел між собою вимагала послідовно 127 операцій сумування. В новітніх процесорах ця операція виконується одною інструкцією [7].

Загалом, CPU виступає як інтегрований показник, що відображає здатність процесора ефективно виконувати обчислювальні завдання. Врахування всіх цих параметрів у зваженій оцінці дозволяє більш точно моделювати вплив апаратної частини на швидкодію програмного додатку.

Memory – компонент системи, який забезпечує доступ до пам'яті для зберігання даних, відображає обсяг наявної оперативної пам'яті, тактову частоту роботи.

Network – частина системи, яка забезпечує комунікацію програмного додатку з іншими пристроями або мережами, включаючи швидкість передачі даних, пропускну здатність та надійність з'єднання.

Задача полягає у визначенні такої комбінації внутрішніх чинників *T, R, P, N, A* та зовнішніх CPU, Memory, Network, щоб функція швидкодії S досягала максимального значення:

$$S^* = f\left(R_*, P_*, N_*, A_*, CPU_*, Memory_*, Network_*\right) = \\ = \underset{R,P,N,T_{arg},D,L\in\Omega}{extr} f\left(R, P, N, A, CPU, Memory, Network\right). \quad (2)$$

Внутрішні чинники мають двосторонні обмеження на множину допустимих розв'язків Ω :

$$R_{\min} \le R \le R_{\max},$$
$$P_{\min} \le P \le P_{\max},$$
$$N_{\min} \le N \le N_{\max},$$
$$A_{\min} \le A \le A_{\max}.$$

$R_{\min}$ та $R_{\max}$ – мінімальна та максимальна кількість операцій введення-виведення відповідно. $R_{\min}$ досягається за відсутності дублювання операцій введення-виведення, коли всі необхідні дані N можуть бути збережені в оперативній пам'яті, а введення-виведення здійснюється з максимальною компресією. $R_{\max}$ спостерігається, коли дані N не можуть бути збережені в оперативній пам'яті, і кожна обчислювальна операція P потребує окремої операції введення даних (наприклад, мережевий запит до бази даних) без застосування компресії.

$P_{\min}$ та $P_{\max}$ – мінімальна та максимальна кількість обчислювальних операцій для розв'язання поставленої задачі. $P_{\min}$ досягається при ефективній математичній реалізації алгоритму з повторним викорис-

танням попередньо розрахованих значень. $P_{\max}$ виникає за відсутності перевикористання попередніх результатів та наявності додаткових операцій, які не впливають на кінцевий результат. Наприклад, використання в хеш-колекціях ключів у вигляді строкового представлення унікального ідентифікатора (*Uniqueidentifier*) замість числового значення значно збільшує кількість операцій.

Для отримання найбільш точних результатів моделювання поведінки системи, необхідно використовувати реальні дані з досліджуваної системи. Наприклад, хеш-алгоритми [8] прискорюють пошук елементів у колекції при перевищенні певного порогу кількості значень. При використанні замалої кількості даних, хеш-алгоритми покажуть гіршу швидкодію за лінійний пошук і помилково можуть бути використані в промислових системах. Це призведе до падіння швидкодії через невірно обраний параметр N. Параметри $N_{\min}$ та $N_{\max}$ виступають як обмеження в досліджуваних системах, визначаючи мінімальний та максимальний обсяг даних яким оперує система.

Параметр N дозволяє врахувати наявність надлишкового використання пам'яті при наявності дублікатів незмінних об'єктів [9], завеликих обсягів пам'яті виділених для невеликого діапазону даних.

Враховуючи, що параметри апаратної частини відомі, та набувають сталих значень в досліджуваних системах, в математичній постановці задачі, подамо їх як константи.

При використанні паралельних обчислень, виникає необхідність додаткових операцій для синхронізації результатів, забезпечення цілісності даних. Наприклад, одночасний запис в наступний вільний елемент масиву призведе до втрати одного зі значень. Для вирішення цієї проблеми використовуються примітиви синхронізації, як мютекси та блоки синхронізації [10], Згідно введених визначень, кількість операцій P буде збільшуватись з урахуванням синхронізації даних та необхідністю управління потоками. Збільшення кількості операцій дозволяє покращити швидкодію: $S_{parallel} > S$.

Виконання паралельних обчислень може покращити швидкодію для алгоритмів, які мають розрахункові блоки не залежні між собою, та час виконання ітерації є значно більшим ніж час для планування потоку виконання [11]. Наприклад, алгоритми створені з послідовних обчислень, не можуть виконуватись швидше паралельно.

З метою урахування цих чинників, введемо функцію оцінки складності алгоритму $T_{alg}(N, P, R)$ в залежності від кількості операцій, можливості паралелізації, необхідності виконання операцій введення-виведення; функцію оцінки використання паралельних обчислень та розподілених систем для прискорення виконання завдань $D(T_{alg}, N, P, R)$; функцію оцінки якості програмного коду, враховуючи оптимізацію

алгоритмів, діапазони оброблюваних даних, патерни використання пам'яті $L(D, T_{alg}, N, P)$.

Для урахування сторонніх факторів, як реалізація системи планування потоків операційною системою, наявність віртуалізації [12] та інші, введено параметр A. Накладено обмеження на множину допустимих розв'язків $A_{\min} \le A \le A_{\max}$, максимальна швидкодія досягається при відсутності сторонніх чинників, які впливають на обчислювальні потужності. Враховуючи виконання обчислювальних експериментів на одних і тих самих фізичних обчислювальних машинах зі сталими версіями операційних систем, в роботі розглядається стале значення змінної A.

До зовнішніх чинників, що описують характеристики апаратної платформи, відносять:

$$CPU \le CPU_{max},\ Memory \le Memory_{max},\ Network \le Network_{max}. \quad (3)$$

Параметри апаратної частини віднесено до детермінованих змінних:

$$CPU \approx CPU_{max},\ Network \approx Network_{max},\ Memory \approx Memory_{max},\ A \approx A_{const}.$$

Описані обмеження визначають множину допустимих розв'язків задачі, та доповнюють математичну постановку задачі, надаючи постановці задачі практичного змісту.

Таким чином, сукупна швидкодія, досліджувана у роботі, визначається як:

$$S = f(R, P, N, T_{alg}, D, L). \quad (4)$$

Метою є знайти таку комбінацію чинників $X = \{T, R, P, ...\}$, що задовольняє умовам та максимізує цільову функцію S.

$$S^* = f\left(R^*, P^*, N^*, T^*_{\text{arg}}, D^*, L^*\right) =$$

$$= \underset{R,P,N,T_{\text{alg}},D,L\in\Omega}{extr} f\left(R, P, N, A, CPU, Memory, Network\right).$$

Знімок пам'яті – це збережений стан оперативної пам'яті програми або системи в певний момент часу, який містить повну інформацію про всі об'єкти, структури даних та їх взаємозв'язки. Аналіз знімків пам'яті дозволяє детально дослідити використання пам'яті програмою, виявити можливі витоки, дублювання даних та неефективні патерни використання пам'яті. Використання цього методу в дослідженні надає можливість точно виміряти обсяг даних N, оцінити складність алгоритму T_{alg} та ефективність програмного коду L, що робить ці параметри детермінованими змінними в математичній моделі.

Метод розв'язку задачі. Для розв'язання задачі максимізації функції швидкодії S за наявності обмежень на внутрішні чинники, можна скористатися методом множників Лагранжа.

Введемо вектор λ множників Лагранжа:

$$\begin{cases} \lambda = \left(\lambda_R, \lambda_P, \lambda_{T_{a\lg}}, \lambda_N, \lambda_D, \lambda_L\right), \\ \Lambda = f\left(C\right) + \sum_{j=1}^{m} \lambda_j \left[g_j\left(C_i\right) - b_j \right], \\ g\left(C_1, ..., C_n\right) = b_j,\ j = 1, ..., m < n. \end{cases}$$

Тоді функція Λ Лагранжа матиме вигляд:

$$\Lambda\left(R, P, T_{\text{alg}}, N, D, \lambda_R, \lambda_P, \lambda_{T_{\text{alg}}}, \lambda_N, \lambda_D, \lambda_L\right) =$$

$$= f\left(R, P, N, T_{\text{alg}}, D, L\right) - \lambda_R\left(R - R_{\min}\right) - \lambda_P\left(P - P_{\min}\right) -$$

$$-\lambda_{T_{\text{alg}}}\left(T_{\text{alg}} - T_{\text{algmin}}\right) - \lambda_N\left(N - N_{\min}\right) - \lambda_D\left(D - D_{\min}\right) - \lambda_L\left(L - L_{\min}\right).$$

Стаціонарні точки функції Λ задовольняють умовам:

$$\begin{cases} \dfrac{\partial \Lambda\left(C_i, \lambda\right)}{\partial C_i} = 0,\ i = 0, ..., n, \\ \dfrac{\partial \Lambda\left(C_i, \lambda\right)}{\partial \lambda_i},\ i = 0, ..., m. \end{cases}$$

Використовуючи визначені змінні досліджуваної системи, отримаємо систему рівнянь:

$$\begin{cases} \dfrac{\partial L}{\partial R} = 0,\ \dfrac{\partial L}{\partial P} = 0,\ \dfrac{\partial L}{\partial T_{\text{alg}}} = 0,\ \dfrac{\partial L}{\partial D} = 0,\ \dfrac{\partial L}{\partial N} = 0, \\ \dfrac{\partial L}{\partial \lambda_R} = 0,\ \dfrac{\partial L}{\partial \lambda_P} = 0,\ \dfrac{\partial L}{\partial \lambda_{T_{\text{alg}}}} = 0,\ \dfrac{\partial L}{\lambda_D} = 0,\ \dfrac{\partial L}{\lambda_L} = 0. \end{cases}$$

Зазначені умови дозволяють визначити оптимальні значення внутрішніх чинників *R*, *P*, T_{alg}, *N*, *D*, *L*. Після знаходження цих значень, стає можливим обчислити максимальну швидкодію $S_{\max}$, підставивши їх у вихідну формулу (2). Значення параметрів визначаються шляхом проведення серії розрахункових експериментів на промислових системах. Враховуючи значний вплив обсягу оброблюваної інформації *N* та кількості операцій *P* на більшість параметрів T_{alg}, *D*, *L*, дослідимо можливості покращення швидкодії через вплив на *N* та *P*.

Оцінка існуючих методів та підходів до оптимізації швидкодії програмного додатку. Кешування даних є ефективним підходом до оптимізації швидкодії. Зберігання попередньо оброблених або часто використовуваних даних у кеші дозволяє швидко отримувати доступ до них, уникнути повторних обчислень і зменшити навантаження на систему. Ефективне використання кешування може значно прискорити виконання програмного додатку. Оцінка впливу кешу-

вання має брати до уваги час доступу до кешу (в пам'яті процесу, або розподілена система як Redis / Couchbase), розміру:

$$\xi = \int_0^1 \left(\tau\rho \frac{C}{D} S \right) dt, \tag{5}$$

ζ – швидкодія після кешування, τ – час доступу до кешу, ρ – ймовірність попадання в кеш, C – розмір кешу, D – розмір даних, S – базова швидкодія без кешування. При проектуванні системи кешування важливо брати до уваги фактичний розмір інформації та розмір технічних метаданих. Наприклад, реалізація хеш-колекції вимагає створення декількох масивів об'єктів та екземплярів допоміжних класів. Фактична пам'ять може бути в десятки разів більшою за розмір інформаційної складової.

Ефективне управління пам'яттю має великий вплив на швидкодію програмного додатку. Використання мінімально необхідного обсягу пам'яті, уникнення витоку пам'яті і оптимізація доступу до пам'яті можуть покращити продуктивність. Використання локальних змінних, ефективне використання кеш-пам'яті і уникання фрагментації пам'яті є важливими аспектами оптимізації роботи з пам'яттю.

$$\Psi = \alpha\Phi + \beta\Theta + \gamma Q, \tag{6}$$

Ψ – ефективність управління пам'яттю, яка враховується при оптимізації. Φ – мінімально необхідний обсяг пам'яті та уникнення витоку пам'яті. Θ – оптимізацію доступу до пам'яті, включаючи ефективне використання кеш-пам'яті. Q – уникання фрагментації пам'яті. *α, β, γ* – вага кожного аспекту в оптимізації, налаштовані залежно від конкретних потреб та пріоритетів. Створені математичні моделі було апробовано на різних промислових системах у сферах електронної комерції, аналітики, обробки банківської інформації, пошукових системах, авіаперевезень, виробництва.

Завдяки серії обчислювальних експериментів на вищезгаданих промислових системах стає можливим узагальнення шаблонів використання пам'яті за допомогою знімків пам'яті. Отримана класифікація дозволяє виявляти помилки при розрахунку розміру об'єктів, збільшене використання пам'яті, та надавати рекомендації.

Використання паралельного програмування дозволяє використовувати багатопотоковість та багатопроцесорні системи для розподілу завдань і прискорення виконання програми. Паралельне виконання обчислень може покращити швидкодію, зокрема для завдань, які можуть бути виконані незалежно. Однак, ефективне використання паралельного програмування вимагає уваги до синхронізації потоків та уникнення гонок даних.

$$\psi = \alpha\Pi + \beta\Sigma + \gamma\Delta, \tag{7}$$

ψ – покращення швидкодії програми після використання паралельного програмування. П – багатопотоковість та багатопроцесорність системи,

що дозволяє розподіляти завдання та прискорювати виконання програми. Σ – ефективність виконання незалежних завдань, які можуть бути розподілені між потоками. Δ – синхронізація потоків та уникнення гонок даних. α, β, γ – вага кожного параметру моделі в оптимізації, налаштовані залежно від конкретних потреб та пріоритетів.

$$0 \leq \alpha \leq 1, 0 \leq \beta \leq 1, 0 \leq \gamma \leq 1.$$

Використання спеціалізованих бібліотек і інструментів, таких як оптимізовані математичні бібліотеки або фреймворки для паралельного програмування, може допомогти покращити швидкодію програмного додатку. Ці бібліотеки та інструменти зазвичай мають оптимізовану реалізацію алгоритмів і надають швидкий доступ до потужних функцій, зменшуючи необхідність вручну виконувати оптимізації.

Бібліотеки, такі як NumPy, Pandas, TensorFlow [13] та PyTorch, дозволяють ефективно обробляти числа, аналізувати та маніпулювати даними, а також навчати глибокі нейронні мережі. Широко застосовувана мова програмування Python має динамічну типізацію, тому для виконання операцій над змінними необхідно спочатку встановити типи цих змінних. Наприклад, для строк операція додавання створить результат який є послідовним з'єднанням двох строк. Для чисел це буде математична сума. Тому реалізація обчислень виключно на Python зменшує швидкодію у десятки [14] разів порівняно з реалізаціями на мові C. Вищезгадані бібліотеки реалізують математичний апарат саме на C, використовуючи Python як декларативний механізм. Бібліотека joblib спрощує паралельні обчислення та розподіл завдань на багатоядерних процесорах. Використання таких спеціалізованих бібліотек та інструментів допомагає покращити ефективність програми та забезпечити швидке виконання обчислень.

Для аналізу знімків пам'яті використовується бібліотека ClrMD, яка дозволяє використовувати C# код для аналізу змісту пам'яті. Прикладом аналізу може бути групування строк для знаходження дублікатів, групування об'єктів в пам'яті за типами.

Програмна реалізація. На основі розроблених рішень, було створено програмний продукт на мові програмування Python. Модуль tkinter – створення графічного інтерфейсу користувача (GUI). Модуль platform – отримання інформації про платформу, на якій запущена програма. Модуль psutil – отримання інформації про системні ресурси, такі як використання CPU, пам'ять, дисковий простір і т.д. Використовується для збору даних про ефективність системи. Модуль sqlite3 – робота з базами даних SQLite. Використовується для збереження даних про час виконання та метрики ефективності. Модуль matplotlib.pyplot – візуалізації даних у вигляді графіків та діаграм. В даному випадку використовується для показу графіків ефективності.

Ці модулі та їх функціональність використовуються для створення програми, яка вимірює час виконання програмного продукту,

обчислює метрики ефективності на основі системних параметрів та відображає результати у вигляді таблиці та графіків.

На початку імпортуються необхідні бібліотеки. Далі створюється з'єднання з базою даних performance_metrics.db та таблиця measurements, якщо вона ще не існує. Ця таблиця буде використовуватися для збереження результатів вимірювань.

Визначаються початкові значення коефіцієнтів, які будуть використовуватися при розрахунку метрик продуктивності. Це коефіцієнти для операцій введення-виведення, складності алгоритму, типу процесора і т.д.

Далі створюється графічний інтерфейс з використанням tkinter. Додаються елементи для введення коефіцієнтів, кнопки для виміру та обчислення.

Функція measure_execution_time() виконує тестову функцію complex_function() і заміряє час її роботи. Потім цей час зберігається в базі даних разом з поточною датою і часом.

Функція calculate_metrics() обчислює різні метрики на основі коефіцієнтів: для операцій ВВ, характеристик процесора, ОС тощо. На їх основі обраховується показник ефективності. Всі дані також зберігаються в базі.

Функції show_measurements() і show_chart() відображають результати вимірювань та будують графік зміни ефективності відповідно. Дані отримуються шляхом запитів до бази даних.

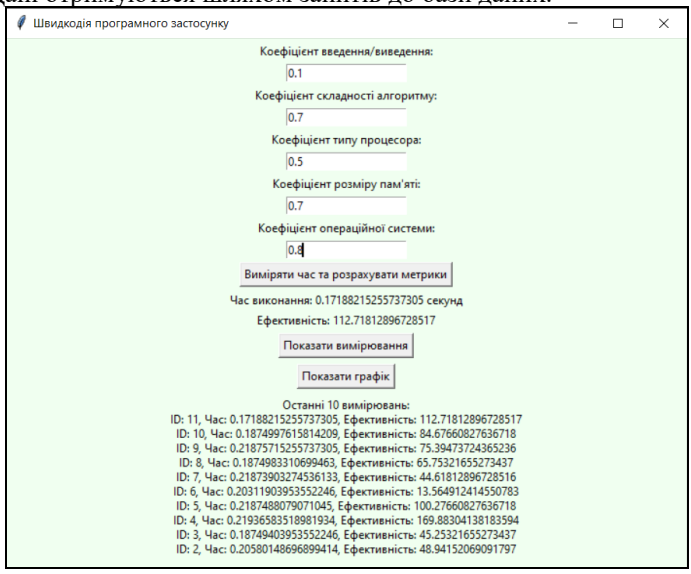

Рис. 1. *Результат роботи програми для визначення швидкодії програмного продукту*

В підсумку програма дозволяє здійснювати вимірювання продуктивності, обчислювати різні метрики та аналізувати їх зміну в часі за допомогою інтерфейсу та графіків.

Аналіз результатів та висновки. Під час експериментального дослідження було виявлено, що швидкодія програмного застосунку, залежить від багатьох факторів, створена математична модель дозволяє описати предметну область.

Дослідження проводилось на промислових системах, які використовують сервера з 36 ядрами процесора, 768 та 512 ГБ оперативної пам’яті.

При пікових навантаженнях, сервера можуть бути збільшені до 1024 ГБ пам’яті та 48 процесорних ядер. Описана математична модель була адаптована до сьогоденних обчислювальних програмних і апаратних реалізацій.

Збільшення обсягу доступної пам'яті позитивно впливає на продуктивність програми. Більший обсяг пам'яті дозволяє зберігати більше даних без необхідності постійного звернення до зовнішніх джерел, що зменшує час доступу до даних і кількість операцій вводу-виводу *R*. Фізичний обсяг оперативної пам’яті має ліміти, тому ключову роль відіграє збереження даних у пам’яті у компактній формі, з можливістю перевикористання незмінних об’єктів [10].

Вибір алгоритму є важливим фактором в ефективності програмного застосунку. Деякі алгоритми можуть бути більш обчислювально витратними, що збільшує час виконання програми. Використання ефективних алгоритмів дозволяє знизити навантаження на процесор і покращити швидкодію програми.

Різні операційні системи мають різні оптимізації та рівень підтримки обладнання і бібліотек, що впливає на швидкодію програми. Наприклад, найбільш популярна бібліотека для машинного навчання TensorFlow не підтримує OS Windows починаючи з 2.10 версії яка була видана наприкінці 2022 року [13].

Час доступу до введених та виведених даних також може впливати на швидкодію програми. Повільне читання або запис даних збільшує час виконання програми.

Згідно розрахункового експерименту виявлено, що найбільший вплив на продуктивність програмного застосунку мають розмір оперативної пам'яті та операційна система. Збільшення обсягу пам'яті дозволяє програмі зберігати більше даних у пам'яті, що зменшує необхідність в постійному доступі до зовнішніх джерел і покращує швидкодію. Враховуючи фізичні обмеження на кількість наявної оперативної пам’яті, особливу важливість набуває раціональне використання наявної пам’яті, а саме зменшення кількості дублікатів, ви-

користання типів даних достатнього розміру. Вибір операційної системи також впливає на продуктивність, оскільки різні операційні системи мають різні рівні оптимізації та підтримки обладнання, що впливає на швидкодію програми.

Щодо інших факторів, таких як складність алгоритму та введення/виведення даних, вони також мають вплив на продуктивність, але менш значущий. Вибір ефективного алгоритму є важливим, адже деякі алгоритми можуть бути більш обчислювально витратними, що збільшує час виконання програми. Однак, під час аналізу виявлено, що ці фактори мають менший вплив порівняно з розміром пам'яті та операційною системою.

У разі введення/виведення даних, швидкодія програми може залежати від швидкості обробки та доступу до введених та виведених даних. Хоча це також впливає на продуктивність, його вплив менш помітний порівняно з розміром пам'яті та операційною системою.

Отже, після аналізу даних можна зробити висновок, що розмір пам'яті та операційна система мають найбільший вплив на продуктивність програмного застосунку, тоді як складність алгоритму та введення/виведення даних мають менший вплив.

Список використаних джерел:

1. Bentley J. L. Writing efficient programs, Englewood Cliffs. N.J.: Prentice-Hall, 1982. ISBN-10 013970244X.
2. Gregg B. 2.7.3 Scaling solutions. *Systems Performance, Second Edition.* Boston: Addison-Wesley. 2021. 929 p. ISBN-10 0136820158.
3. Форкун Ю., Форкун І., Яшина О., Праворська Н. Архітектурні методи оптимізації швидкодії та відмовостійкості програмних застосунків. *ВОТТП.* 2023. С. 196-201.
4. Слабінога М. О., Чабан С. В. Розробка вебдодатків в контексті оптимізації їх швидкодії. *Таврійський науковий вісник. Серія: Технічні науки*. 2022. Вип. 3. С. 63-69. URL: https://doi.org/10.32851/tnv-tech.2022.3.7.
5. Стягар А. А. Продуктивність вебдодатків. *Інституційний репозитарій ТНТУ імені Івана Пулюя: Домівка*. URL: https://elartu.tntu.edu.ua/bitstream/-lib/23917/2/V-STC-IMST_2018_Stiahar_A-Web_applications_perfomance_82.pdf.
6. Ioannis S. E., Christou T. To pool or not to pool? Revisiting an Old Pattern. *Athens Information Technology.* 2018. URL: https://doi.org/10.48550/arXiv.1801.03763.
7. Hossein Amiri, Asadollah Shahbahrami. SIMD programming using Intel vector extensions. *Journal of Parallel and Distributed Computing*. 2020. Vol. 135. P. 83-100. ISSN 0743-7315. URL: https://doi.org/10.1016/j.jpdc.2019.09.012.
8. Ma Y., Xia L., Lin J., Jing J., Liu Z., Yu X. Hardware Performance Optimization and Evaluation of SM3 Hash Algorithm on FPGA. *Information and Communications Security. ICICS 2012. Lecture Notes in Computer Science.* Springer, Berlin, Heidelberg, 2012. Vol. 7618. URL: https://doi.org/10.1007/978-3-642-34129-8_10.

9. Nixon B. A. Management of performance requirements for information systems. *IEEE Transactions on Software Engineering*. 2000. Vol. 26. No 12. P. 1122-1146. DOI: 10.1109/32.888627.
10. Albinali H., Alharbi M., Alharbi R., Aljabri M. Synchronization Techniques for Multi-threaded Web Server: A Comparative Study. *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*. Al-Khobar, Saudi Arabia, 2022. P. 373-379. DOI: 10.1109/CICN56167.2022.10008307.
11. Ong B. W., Schroder J. B. Applications of time parallelization. *Comput. Visual Sci.* 2020. Vol. 23 11. URL: https://doi.org/10.1007/s00791-020-00331-4.
12. Gouda K., et al. Virtualization approaches in cloud computing. *International Journal of Computer Trends and Technology (IJCTT)*. 2014. Vol. 12.4. P. 161-166
13. Install TensorFlow with pip, Tensorflow documentation, as of 2024-09-06. URL: https://www.tensorflow.org/install/pip?hl=en#windows-native.
14. Charles E. Leiserson. 6.172 Performance Engineering of Software Systems, Lecture 1: Introduction and Matrix Multiplication. 2018 MIT. URL: https://ocw.mit.edu/courses/6-172-performance-engineering-of-software-systems-fall-2018/resources/mit6_172f18_lec1.
15. Mitikov N. Y., Guk, N. A. Modeling and automation of the process for detecting duplicate objects in memory snapshots. *Вісник сучасних інформаційних технологій.* 2024. Вип. 7 (2). С. 147-157. URL: https://doi.org/10.15276/hait.07.2024.10.

INVESTIGATION OF SOFTWARE APPLICATION PERFORMANCE ISSUES

The article presents a study of methods for optimizing the performance of software applications aimed at identifying the most effective combination of internal and external factors that maximize the objective function. A generalized mathematical model is described, which includes the main factors affecting performance, such as computation time, the number of input/output operations, the number of computational operations, algorithm complexity, the volume of data processed, the use of parallelism, the architecture of hardware and software platforms, and code efficiency. The importance of using specialized libraries and tools to accelerate computational processes, which is critically important for achieving high performance in modern software systems, is emphasized.

The developed approaches were implemented in software, allowing for the practical evaluation of the proposed methods. Software modules were created to analyze the impact of various factors on performance, considering the specifics of particular tasks and execution environments. The test results demonstrated significant potential for performance improvement through optimization at both the code level and the hardware architecture level.

Particular attention is given to the study of memory management, addressing potential challenges that negatively impact performance. The necessity of using a caching system and avoiding duplication of immutable information is highlighted. The identified scenarios are independent of any

specific implementation and can therefore be integrated into the developing recommendation system.

The research has practical significance, offering comprehensive solutions for optimizing the performance of software systems that can be applied in industrial high-load environments. Further research will focus on expanding the functionality of the recommendation system, integrating more complex optimization models, and conducting large-scale computational experiments to validate the results under real-world conditions.

Keywords: *performance, software applications, mathematical models, optimization methods, software implementation, recommendation system, decision tree.*

Отримано: 26.08.2024

УДК 536.2
DOI: 10.32626/2308-5916.2024-25.36-45

Р. С. Мусій, д-р. фіз.-мат. наук, професор,
М. І. Клапчук, канд. фіз.-мат. наук,
А. В. Кунинець, канд. фіз.-мат. наук,
І. Г. Свідрак, канд. техн. наук,
Р. Я. Пелех, аспірант

Національний університет «Львівська політехніка», м. Львів

МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ ТА КОМП'ЮТЕРНИЙ АНАЛІЗ ТЕПЛОВИХ РЕЖИМІВ ЕЛЕКТРОПРОВІДНОЇ ПАНЕЛІ ПРИ ІНДУКЦІЙНІЙ ТЕРМООБРОБЦІ

Розглядається електропровідна панель прямокутного поперечного перерізу. За індукційної термообробки квазіусталеним електромагнітним полем у ній виникають нестаціонарні об'ємно розподілені джерела тепла Джоуля. При відповідних параметрах зовнішнього електромагнітного поля теплові процеси в панелі можуть відбуватися в умовах приповерхневого або суцільного нагріву. Для дослідження закономірностей її теплових режимів за вищезазначених умов нагріву запропоновано двовимірну фізико-математичну модель. Дана модель складається з двох етапів. На першому етапі зі співвідношень Максвелла визначається дотична до основ панелі компонента вектора напруженості магнітного поля. На другому етапі за знайденою цією компонентою знаходиться тепло Джоуля. Для побудови розв'язку задачі електродинаміки використано апроксимацію розподілу дотичної до основ панелі компоненти вектора напруженості магнітного поля по товщинній координаті кубічним поліномом. Коефіцієнти апроксимаційного полінома подаються у вигляді лінійної комбінації інтегральних за тов-